Learning Features that Predict Developer Responses for iOS App Store Reviews

Kamonphop Srisopha
University of Southern California
Los Angeles, California
srisopha@usc.edu

Daniel Link
University of Southern California
Los Angeles, California
dlink@usc.edu

Devendra Swami
University of Southern California
Los Angeles, California
dswami@usc.edu

Barry Boehm
University of Southern California
Los Angeles, California
boehm@usc.edu

ABSTRACT

Which aspects of an iOS App Store user review motivate developers to respond? Numerous studies have been conducted to extract useful information from reviews, but limited effort has been expended to answer this question. To address this, we study the relationship between a wide range of features in reviews and whether they received a response from the developers. For our prediction, we run a random forest algorithm over the derived features. We then perform a feature importance analysis to understand the relative importance of each individual feature and groups thereof. Through a case study of eight popular apps, we show patterns in developers' response behavior. Our results demonstrate not only that rating and review length are among the most important but also that review posted time, sentiment, and the writing style play an important role in the response prediction. Additionally, the variation in feature importance ranking implies that different app developers use different feature weights when prioritizing responses. Our results may provide guidance for those building review or response prioritization tools and developers wishing to prioritize their responses effectively.

CCS CONCEPTS

• **Information systems** → **Data mining**; **Retrieval models and ranking**; • **Software and its engineering** → *Software evolution.*

KEYWORDS

Developer Response, Prioritization, User Reviews, Feature Importance, App Store Mining, Random Forest

1 INTRODUCTION

The ever-tightening web of social interactions has laid the groundwork for closer communications between the developers and users of applications. In 2017, the iOS App Store has, similar to many other vendors, taken advantage of this to empower both sides through adding the option of following up on reviews with responses such as advice or clarifications[1].

While users and developers are positioned at the opposite ends of the exchange of information, both are interested in optimizing their input/output ratio. Consequently, users would like to gain the attention of the developers and receive a response that addresses their concerns; developers value having their app ratings raised after posting an explanation or, possibly, a note that the app has been updated based on the feedback. This requires developers to show a commitment to listening to the community of users and making improvements accordingly. Since typically, their resources, such as time and available personnel, are limited, they have to prioritize their responses, as evident by the fact that a small fraction of reviews received a response [9]. Nonetheless, responding to reviews can improve overall rating and user satisfaction [9, 18].

From the users' perspective, a major point of interest is how to shape and place a review so that it garners a response from the developers. Previously, there have been efforts made toward understanding the nature of how users review apps and different approaches that automatically classify and summarize reviews to help developers accelerate software maintenance and evolution tasks [17]. On the other hand, less effort has been expended in studying which features out of many in an app review prompt a developer's response. Understanding the key features that influence developers' response decision could provide guidance for those building review or response prioritization tools and meaningful direction to developers wishing to prioritize their responses effectively. Generally, better response strategies could lead to increased user satisfaction and improved communication between users and developers.

Motivated by the above observations, we investigate a wide range of features that can be extracted from user reviews and apply a random forest to study how these features contribute to whether developers will respond to a review. Through a case study of eight popular free-to-download apps, the results show that with the identified features as input, the random forest algorithm is able to differentiate between reviews that receive a developer response from those that do not, outperforming all baselines with large gains. This outcome suggests that there exist patterns in developers' response behavior. Additionally, we perform a feature importance analysis to understand the relative importance of each individual feature and group of features in predicting a developer response. Insights discovered in this paper could be of practical significance to broad range of practitioners.

Our main contributions can be summarized as follows:

- We investigate a wide range of features that can be extracted from user reviews on the iOS App Store that are likely to be associated with developers' response behavior. Different

[1] https://developer.apple.com/library/archive/releasenotes/General/WhatsNewIniOS/Articles/iOS10_3.html

from previous work, we study many novel features such as writing style, vote, time, and intention.
- We extract 7 different groups of features and build models to predict developer responses. We identify the relative importance of each individual feature and groups of features. We provide insights into the relationships between the key features and developer responses.
- We demonstrate a potential application to apply feature engineering and machine learning to aid in the automatic prioritization of developer responses or user reviews.

We organize the remaining of the paper as follows. In Section 2, we present background information. In Section 3, we describe our research setup. In Section 4, we present the experiment results. In Section 5, we discuss the relationship between the key features and developer responses. In Section 6, we explore implications. In Section 7, we identify threats to validity and limitations. In Section 8, we give an overview of the related literature. In Section 9, we conclude the paper and outline future directions.

2 BACKGROUND

2.1 Definitions and Examples

In this paper, the term **user** refers to the end user of an app. Additionally, **developer reaction** refers to what the developer(s) of an app will do after seeing a user's review. They could (1) do nothing, (2) write a public response on the app store, or (3) update the app based on cues taken from the review without communicating those changes to the users, or both of the former options. In this study, we focus only on the second type of reaction, i.e., an instance where developers provide a written response to a review on the App Store. We call this **a developer response**.

Figure 1 shows three examples of actual user reviews that received a developer response. In the review example on the left, a reviewer reported an unexpected behavior of the app and gave a one-star rating. Five days later he received a developer response providing troubleshooting steps to solve his issue. In the review example on the top right, a reviewer suggested developers to add a new feature. Shortly after, a developer responded that an identical feature had already been implemented in the version of the app. Lastly, we chose the review example on the bottom right to show an instance where a user edits his review after receiving a developer response. The original content of the review is unknown as it was replaced by the edited version. In such a case, the *edited* indicator will appear next to the posted date.

2.2 Review and Response Mechanism

After users download an app from the iOS App Store, they can share their experience and opinion about the app by leaving a public review, consisting of a text review and a star rating. The star rating ranges on a scale of one to five. Users cannot write multiple reviews for an app, but they can edit or delete their reviews and ratings. Users can also upvote or downvote reviews they find helpful or not helpful, respectively.

Developers, on the other hand, see all reviews written by users through the app management portal. They can filter reviews based on, for example, their ratings, countries of origin, or versions of the app. They can also sort reviews, for example, by helpfulness

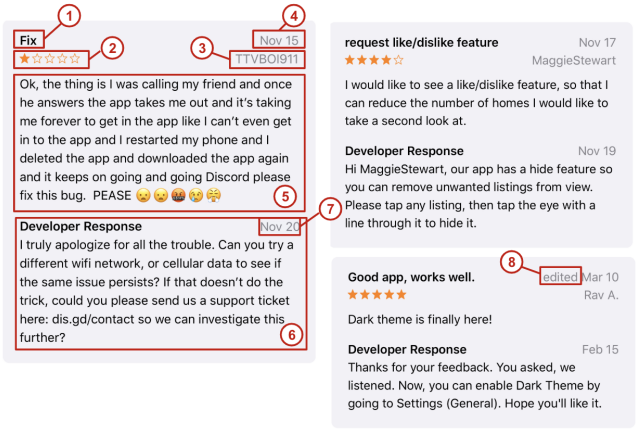


Figure 1: User reviews with a developer response. (1) the title; (2) the star rating; (3) the name of the user; (4) the review posted date; (5) the body; (6) the developer response block; (7) the response posted date; (8) the edited indicator.

(the most upvoted reviews). They cannot delete reviews, but they can give public responses to them. Each review can only have one response associated with it. If developers respond to a review, an email, including an option to edit the review, is sent out to the respective user.

There is no limit for the length of reviews and responses users and developers can write, respectively. Once users and developers edit their reviews or responses, the current reviews or responses will be replaced by the edited version.

3 RESEARCH METHODOLOGY

3.1 Task and Research Questions

The aim of this work is to investigate the potential of using a machine learning algorithm and the features that can be extracted from user reviews to model developers' response behavior. While doing so, we want to uncover the learned relationship between these features and developer responses. Our overarching goal is to provide insights into the subject or guidance to those building review or response prioritization tools. We formally define our task as follows.

Task: Developer Response Prediction

- **Given** a review R.
- **Predict** whether developers will provide a response to R.

We then set out to answer the following research questions.

- **RQ1**: How effective is the random forest algorithm at predicting developer responses to user reviews?
- **RQ2**: Which individual features and groups of features are most important in predicting developer responses?

3.2 Features

In this section, we present the features we considered in this study. The choice of the features we selected are mainly influenced by:

(1) Prior work – Rather than starting from scratch, we chose to leverage the wealth of knowledge from prior work to focus on features that have been shown to associate with response decisions in other areas of study (e.g., in Q&A forums [19] and in email communications [34]).
(2) The availability of the features and the collection method – Due to the unstructured and informal nature of user reviews, some features are not trivial to extract. Hence, we focus on features that can be extracted easily or by using tools that have been employed successfully in literature.

Table 1 shows the summary of features considered in this study. We categorized our selected features into 7 groups based on the characteristics of a feature within the group. Group 1 to 3 are non-text features and group 4-7 are text features. We believe that the short descriptions in Table 1 are explanatory enough to some of the features. Due to the limited space, we will only discuss features that deserve a longer explanation.

3.2.1 ***Time.*** In email communications, the sent date and time of emails affects email reply behavior [34]. The posted date and time of reviews may also affect developer response decisions as developers may be more active during a certain time of the day and on weekdays. In addition, Hassan et al. [9] found evidence suggesting that developers responded mostly to reviews which are posted shortly after a release. Hence, in this group, we consider *TimeOfDay*, *DayOfWeek*, *IsWeekEnd*, and *TimeAfterRelease*.

Following Yang et al. [34], we partitioned the time of day into four sections: night ($0/24-6$), morning ($6-12$), afternoon ($12-18$), and evening ($18-24/0$).

3.2.2 ***Style.*** Several features of the writing style of the reviewer were found to influence the perceived value of reviews [4, 13, 27]. Hence, they may also affect developers' perception of a review and thus affect their reaction.

We used the textstat[2] package to calculate the Flesch's reading ease (readability) score [5]. To detect capital letters, we used the built-in string function provided directly by Python. For the rest of the features in this group, we applied common pre-processing steps to reviews. Specifically, we converted them to lower case and used the Treebank tokenizer[3] to split a particular review into words or tokens. Note that some style features may call for additional steps which will be discussed below. Furthermore, we obtained the proportion of each feature by normalizing the count of it by the number of words in the review.

To detect spelling errors, we checked each word in a review against an English dictionary[4]. To count modal verbs, we matched each word against the common modal verbs: *can*, *could*, *may*, *might*, *must*, *will*, *would*, *shall*, *should*, and *ought*. We also included their negative and contraction forms. To count unique words in reviews we added the widely adopted Porter Stemmer algorithm [24] to the pre-processing steps. This is done to reduce complexity and semantic duplicates in reviews by transforming the word back to its root form (*shopping* to *shop*). Finally, to derive the ContainsDigits feature, we checked each character in a review to see if it is a digit.

3.2.3 ***Sentiment.*** Studies have shown that sentiment can help indicate the user's intention for writing review [15, 23]. Additionally, developers may want to prioritize responding to negative reviews since such reviews are likely to result in increased user satisfaction. Thus, we consider sentiment as an important feature in our study. We use *SentiStrength* [32], a lexicon based sentiment analysis technique, to calculate the sentiment. It is chosen due of its wide adoption in several mobile app user reviews studies [7, 9, 15]. Lastly, trinary sentiment classification is chosen to report as SentiStrength output, which assigns positive, negative, or neutral sentiment to reviews.

Additionally, Salehan et al. [25] found correlation between the amount of positive and negative words present in reviews with whether a review will be read. To identify positive and negative words in reviews, we used the opinion lexicon corpus by Hu and Liu [11]. We then normalized the count of each type by the number of words in the review to account for variability in review lengths.

3.2.4 ***Intention.*** Each review can consist of multiple sentences, each with its own intention. Some intentions are actionable, while the others may not be. Panichella et al. [23] grouped 17 common topics present in user reviews, as identified by Pagano et al. [20], into four intentions (listed in group #7 of Table 1) that are relevant to developers.

We used the *ARdoc* tool, a user review classifier by Panichella et al. [23], to detect intention in user reviews. The tool assigns each sentence in a review one of the four intentions. We chose the tool because it categorizes reviews at a sentence-level and has been used by other tools, such as in *ChangeAdvisor* [22] and in *SURF* [3]. We directly used the original Java implementation of the tool[5].

3.3 Selected Apps

We obtained a list of 50 popular free-to-download iOS apps in the US based on the ranking by AppAnnie[6], a market research company which monitors app downloads in the iOS App Store. For each app on the list, we used a web crawling tool we developed to retrieve all available reviews by US-based users that were posted within a period from October 1st, 2018 to March 2nd, 2020 (UTC). The crawler acts like an iOS device and utilizes the iTunes APIs to collect data about an app. The length of our study period should allow enough time for these apps to go through many releases and gather user reviews and developer responses. The decision to select only popular apps is based on two reasons. First, such apps have more users than unpopular ones and should therefore have more user review data to analyze. Second, we believe that their developers care about maintaining their popularity by motivating their users to keep using them, which requires good support including responses.

We have selected eight apps from the list for our case studies under the consideration that response rates[7] that are too high ($> 50\%$) or too low ($< 5\%$) suggest that their developers are inelastic in their response rates. Table 2 summarizes the information, including the name, category, number of reviews, number of developer responses, number of edited reviews, and response rate for each app in our

[2]https://pypi.org/project/textstat/
[3]https://www.nltk.org/_modules/nltk/tokenize/treebank.html
[4]https://github.com/dwyl/english-words
[5]https://www.ifi.uzh.ch/en/seal/people/panichella/tools/ARdoc.html
[6]https://www.appannie.com/en/
[7]The response rate is the percentage of all reviews with a developer response over all reviews.

Table 1: Review features potentially affect developer responses along two dimensions: Non-text (#1-#3) and Text (#4-#7)

#	Group	Feature	Description
1	Time	TimeOfDay	The time of the day that the review is posted.
		DayOfWeek	The day of week that the review is posted.
		IsWeekend	Whether the posted date is a weekday or a weekend.
		TimeAfterRelease	The time, in hour, between the release of a version and the submission of review for the version.
2	Rating	Rating	The number of stars in the rating.
3	Vote	VoteHelpful	The number of users who vote the review as helpful to them.
		VoteTotal	The total number of votes the review receives.
4	Style†	Readability	The score obtained from Flesch reading-ease test [5].
		MisspelledWords	The proportion of spelling errors in the text.
		ModalVerbs	The proportion of modal verbs in the text.
		UniqueWords	The proportion of unique words in the text.
		ContainsDigits	Whether the text contain any numerical values.
		FirstCap	Whether the text begins with a capital letter.
		AllCap	Whether the text is written in all capital letters.
5	Length†	NumChars	The number of characters in the review.
6	Sentiment†	Sentiment	The overall sentiment polarity of the review (positive, negative, or neutral)
		PositiveWords	The proportion of positive sentiment words in the text.
		NegativeWords	The proportion of negative sentiment words in the text.
7	Intention†	ProblemDiscovery (PD)	Whether the review contains a sentence describing issues or unexpected behaviors of the app.
		FeatureRequest (FR)	Whether the review contains a sentence expressing ideas or suggestions for app improvement.
		InformationSeeking (IS)	Whether the review contains a sentence inquiring information or help from developers.
		InformationGiving (IG)	Whether the review contains a sentence informing developers about an aspect of the app.

†: Applicable to both review title and body.

case study. Note that for our analysis, we have discarded reviews that were edited because we did not have their original contents.

Table 2: Characteristics of selected apps

App Name	Category	Reviews	Responses	Edited	Rate
Bank of America	Finance	14754	2833	245	19.2%
Discord	Social Network	8707	1837	186	21.10%
Mercari	Shopping	9047	1540	261	17.02%
PayPal	Finance	19597	2163	76	11.03%
Realtor	Lifestyle	18779	6508	25	32.66%
Reflectly	Fitness	17010	1435	83	8.44%
StockX	Shopping	16805	4536	324	26.99%
WeatherBug	Weather	8183	2320	180	28.35%

3.4 Classification Algorithm

Our goal for building a machine learning model is to use the model to gain insight on the subject through examining the learned relationship between the features and developer responses.

With this goal in mind, we experimented with the random forest algorithm [1], one of the most successful non-linear machine learning algorithm, suitable for this type of task. The random forest algorithm is an ensemble algorithm made up of multiple decision trees. Each tree in the forest is trained on a random sample of the trained data drawn with replacement (known as bootstrapping). It also randomly selects subsets of features when splitting nodes and makes the decision by averaging the predictions from each decision tree. As a result, the random forest algorithm is less prone to overfitting. In addition, the use of an internal error estimation through Out-of-Bag samples removes the need of a set-aside validation or test set. The algorithm is also flexible as it can take numerical or categorical variables as input and does not require feature scaling. For these reasons, we chose the random forest algorithm for our study.

We have used Scikit-learn[8], a machine learning library for Python, to implement the random forest algorithm. The dataset of each app is split into 80% training and 20% test data using *stratified* sampling to ensure similar label distribution across test and train sets.

Hyperparameters optimization: We have constructed a grid search space and performed randomized search over uniformly drawn samples from it. The range of values used for different hyperparameters to create this grid search space are as follows:

- **n_estimators**: 100 to 1000 with step size of 100.
- **max_features**: integral multiple of $\sqrt{\#features}$ to #features.
- **max_depth**: 10 to 100 with step size of 10.
- **min_samples_split**: 0.01, 0.03, and 0.05.
- **min_samples_leaf**: 1, 3, and 5.
- **bootstrap**: always set to *True*.
- **class_weights**: always set to *balanced*.

The above set of values leads to a grid search space of 5400 ($10 \cdot 6 \cdot 10 \cdot 3 \cdot 3$). We then performed random sampling using *RandomizedSearchCV* (from scikit-learn) to try out a wide spectrum of values from the above search space. We have used 100 iterations

[8] https://scikit-learn.org/

through 4−fold cross validation to identify the optimal hyperparameters. We report the hyperparameters obtained from the procedure for each app in Table 3.

Table 3: Chosen hyperparameters for each app using RandomizedSearchCV.

Parameter	Bank of America	Discord	Mercari	Paypal
n_estimators	100	1000	500	500
max_features	41	42	34	34
max_depth	90	20	20	20
min_samples_split	0.01	0.01	0.01	0.01
min_samples_leaf	3	1	1	1

Parameter	Realtor	Reflectly	StockX	WeatherBug
n_estimators	900	700	400	600
max_features	20	6	34	34
max_depth	10	40	40	90
min_samples_splits	0.01	0.01	0.01	0.01
min_samples_leaf	3	1	3	3

3.5 Baselines

To evaluate the effectiveness, we compare the performance of the random forest algorithm described earlier to the following two standard baseline approaches.

- **Random**, we randomly assign a class to each review in the test set.
- **MajorityClass**, we assign the majority class in the training set to the testing set.

Generally, if the performance of our model is too close to these baselines, it implies that the selected features have no association with developer responses or more features are needed.

3.6 Performance Metrics

Table 2 shows that our dataset is imbalanced, with the majority of reviews not receiving a developer response. The average response rate for all case study apps is 20%. Therefore, for our task, *Accuracy* is not a suitable evaluation metric. That is because we can predict that developers will not respond to any reviews and achieve an accuracy of 80%, but we gain no knowledge on our research subject.

Instead, we select the Area Under the Receiver Operating Characteristic (ROC) Curve to evaluate the performance on our task. We use the acronym *AUC* to denote this metric. The ROC curve shows how the true positive rate and false positive rate relationship changes as the threshold for identifying positives in the model changes. The area under this curve (AUC) indicates the performance of a model at separating classes. A perfect classification model would give an AUC value of 1.00, while a random model would give an AUC value of around 0.50. Generally, the higher the AUC value, the better the model.

3.7 Feature Importance Analysis

We employed a model-agnostic technique called "permutation feature importance" which was introduced by Breiman [1] to measure the importance of an individual feature and a group of features.

The key idea of this technique is if a random permutation of a feature leads to a substantial increase in the classification error, the model heavily relies on that particular feature for prediction. We refer to this increase as the **Importance Value (IV)**. Hence, the higher the increase in the classification error, the more important the feature is to the model. We used the training data to compute importance. By using this same principle, we can compute the importance of a group of features by permuting features in the same group together (i.e., as a single meta-feature). The increase in the classification error is the relative importance for that group.

Nonetheless, Strobl et al. [31] noted that when two or more individual features are highly correlated, the technique can lead to an incorrect identification of feature importance due to the fact that the model can use the same information from a correlated feature. Hence, one strategy to mitigate this is to identify and discard highly correlated features prior to training the model. For this, Spearman's correlation is used to identify correlated features, due to its ability to model complex monotonic relationships among features rather than just linear as modeled by Pearson correlation method. By following Omondiagbe et al. [19], feature pairs with absolute correlation value of 0.7 or higher are marked as highly correlated. Among these identified pairs, the ones with a lower mutual information [14] are subsequently removed. This approach ensures that our models are trained on features which are not significantly correlated with one another. Consequently, we can safely employ the permutation feature importance technique.

4 RESULTS

4.1 RQ1: How effective is the random forest algorithm at predicting developer responses to user reviews?

Figure 2 shows the ROC curves of the baselines and the random forest algorithm which uses the identified features as input for each case study app. We observe that the random forest algorithms that use the identified features as input performed significantly better than the two baselines at distinguishing reviews that received a developer response from those that did not, for all eight apps. The average AUC for the random forest algorithms is 0.84. Regarding the two baselines, since the *MajorityClass* baseline always predicts 'No' or 0, its true positive rate is 1.0 because reviews with no developer responses were correctly classified, and the false positive rate is 1.0 since reviews with developer responses were incorrectly classified, thus achieving an average AUC score of 0.500 across all eight apps. Similarly, *Random* achieved expected AUC scores of around 0.500 due to their randomness.

For the app *WeatherBug*, the model is able to achieve an AUC score of 0.99, which is considered almost perfect. This indicates that we have identified features that developers for this app use to prioritize their responses. However, for the app *StockX*, the random forest is not able to achieve as high AUC score as other apps, although higher than the two baselines. This may indicate that we may still be missing crucial features that developers for this app use to decide whether a review should get a response or that developers do not have a concrete response strategy.

Overall, the high average AUC score of 0.84 indicates that the models have a high explanatory power in predicting developer

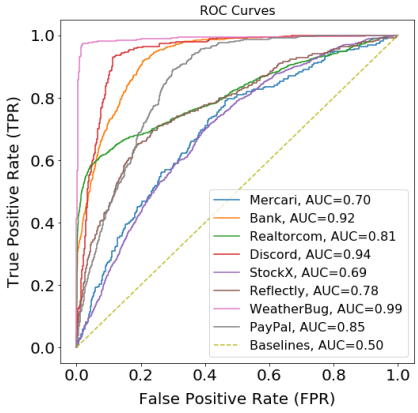


Figure 2: The ROC curves of the random forest algorithm that uses the selected features for the baselines and each case study app.

responses and that the selected features have some association with developer responses. This suggests that an application which applies feature engineering and machine learning would have the potential to help developers automatically identify reviews they are most likely to respond to.

4.2 RQ2: Which individual features and groups of features are most important in predicting developer responses?

Table 4 shows the top 10 most important features in predicting developer responses for each app. We also observe that *Rating* is among the most important features across most apps, with the exception of the apps *StockX* and *Reflectly*. This is expected since the rating is an important, at-a-glance representation on how users feel about an app. Developers can quickly gauge how satisfied their users are with an app just by looking at the rating and without having to read the review contents. This result is congruent with that of previous work of Hassan et al. [9] and McIlroy et al. [18], which found the rating to play some role in deciding whether a developer will respond to a review. However, they did not perform response prediction and feature importance analysis on these features.

Another feature that appears within the top five most important features of most apps is *NumCharsBody*, the feature that has consistently found to be positively correlated with the perceived value of a text [20, 27]. Here, we also observed that the random forest algorithms relied on this feature for both review title and body to predict developer responses.

For the *Sentiment* feature group, *PositiveWords* plays significant roles in predicting developer responses as they consistently appear within the top five most important features for most apps. This may indicate that a review that appears to be overly emotional or in a neutral tone can incite developer responses. Interestingly, our results suggest that *SentimentBody* and *SentimentTitle* are less important for prediction than other features in the same group.

The *Time* features are also important in predicting developer responses as *TimeAfterRelease* appears within the top three most important features for developer response prediction for all apps. Likewise, *DayOfWeek* features (such as *IsThursday*) and *TimeOfDay* features (such as *IsMorning*) can also help with the prediction. The results strongly suggest that the posted date and time of review can increase the likelihood of it getting a developer response.

In the case of the *Intention* feature group, we observe that, for four apps (*Discord*, *PayPal*, *Realtor*, and *WeatherBug*), the random forest algorithms relied on the *Problem Discovery* feature for review body to help predict developer responses. This suggests that developers for these apps often respond to reviews that report unexpected behaviors of the apps. Interestingly, the *FeatureRequest* feature for review body is important for the developer responses prediction of only two apps (*Realtor* and *WeatherBug*). The *Information Giving* and the *Information Seeking* features, despite the latter being more actionable, provide a weak contribution to the prediction. We note that the intentions for the review title also provide a weak contribution to the prediction. Only the app *Discord* has the *Problem Discovery* feature for review title in its most important features list.

Although not all *Style* features appear in Table 4 for most apps, some *Style* features such as *Readability*, *MisspelledWords*, *ModalVerbs*, and *UniqueWords* did play some role in predicting developer responses. This suggests that the quality of reviews does matter to developers to some extent. Interestingly, whether a review contains a numeric value (*ContainsDigits*), is written in all capital letters (*AllCap*) or begins with a capital letter (*FirstCap*) does not contribute to the prediction. Similarly, in all apps, the random forest algorithms did not rely on the *Vote* feature group (*VoteHelpful* and *VoteTotal*).

By investigating the results of the feature group analysis in Table 5, we observe that for most apps, *Rating* is still the most important feature. However, for three apps (*Mercari*, *Reflectly*, and *StockX*), the *Time* feature group is more important than *Rating*. Interestingly, the *Intention* group and *Vote* group are among the least important feature groups in predicting developer response for most apps. These results are also consistent with the results in the individual feature analysis from earlier. By grouping features into *Text* and *Non-text* features, we find that *Non-text* features are more important than *Text* features for the task for all almost all apps, except the app *StockX*. However, the importance value between *Text* and *Non-text* for the app indicate that both groups of features are equally important to the prediction.

Most importantly, Table 4 and Table 5 show some variation in the ranking of features which indicates that the developers of these apps have different strategies for responding to reviews. In the next section, we explore the direction of relationship between important features and developer responses.

5 INSIGHTS

While feature importance analysis (in RQ2) can reveal the relative importance of each individual feature in predicting developer responses, it does not determine the direction of influence or the relationship between the features and developer responses. In this section, we explore the relationship between the key features and developer responses and provide guidance for those building review or response prioritization tools. However, we note that this is a

Table 4: Top 10 most important features in predicting developer responses for each app. (IV = Importance Value)

Bank of America		**Discord**		**Mercari**		**Paypal**	
Feature	IV	Feature	IV	Feature	IV	Feature	IV
Rating	0.179	Rating	0.350	Rating	0.183	Rating	0.157
TimeAfterRelease	0.049	NegativeWordsBody	0.009	TimeAfterRelease	0.160	PositiveWordsBody	0.048
PositiveWordsBody	0.011	TimeAfterRelease	0.009	NumCharsBody	0.079	TimeAfterRelease	0.035
NegativeWordsTitle	0.009	PositiveWordsBody	0.008	PositiveWordsBody	0.052	MisspelledWordsBody	0.029
IsWeekend	0.008	NumBodyChars	0.007	NumCharsTitle	0.038	PDBody	0.027
NumCharsBody	0.007	PositiveWordsTitle	0.007	NegativeWordsBody	0.033	NumCharsBody	0.025
IsMorning	0.007	MisspelledWordsBody	0.005	PositiveWordsTitle	0.029	NegativeWordsBody	0.010
IsWednesday	0.007	IsWeekend	0.005	MisspelledWordsBody	0.022	ModalVerbsBody	0.009
IsTuesday	0.004	PDTitle	0.004	ModalVerbsBody	0.022	NumCharsTitle	0.007
ModalVerbsBody	0.003	NumCharsTitle	0.003	ReadabilityBody	0.018	PositiveWordsTitle	0.006

Realtor		**Reflectly**		**StockX**		**WeatherBug**	
Feature	IV	Feature	IV	Feature	IV	Feature	IV
Rating	0.163	TimeAfterRelease	0.154	NumCharsBody	0.162	Rating	0.282
NumCharsBody	0.032	Rating	0.050	TimeAfterRelease	0.160	TimeAfterRelease	0.007
TimeAfterRelease	0.020	PositiveWordsBody	0.038	PositiveWordsBody	0.023	IsThursday	0.002
PositiveWordsBody	0.018	IsTuesday	0.030	IsWeekend	0.021	PositiveWordsBody	0.001
NegativeWordsBody	0.010	NumCharsBody	0.029	NumCharsTitle	0.014	PDBody	0.000
NumCharsTitle	0.009	NumCharsTitle	0.021	IsSaturday	0.014	FRBody	0.000
ModalVerbsBody	0.008	NegativeWordsBody	0.021	UniqueWordsBody	0.013	ISBody	0.000
FRBody	0.007	IsThursday	0.021	IsFriday	0.010	IGBody	0.000
PDBody	0.005	ReadabilityBody	0.017	MisspelledWordsBody	0.007	NumCharsTitle	0.000
PositiveWordsTitle	0.005	PositiveWordsTitle	0.016	PositiveWordsTitle	0.007	SentimentBody	0.000

Table 5: The relative importance of each group of features in response prediction. (IV = Importance Value)

Bank of America		**Discord**		**Mercari**		**Paypal**	
Feature	IV	Feature	IV	Feature	IV	Feature	IV
Rating	0.179	Rating	0.350	Time	0.184	Rating	0.157
Time	0.088	Sentiment	0.031	Rating	0.183	Sentiment	0.081
Sentiment	0.035	Time	0.018	Sentiment	0.133	Style	0.045
Style	0.012	Style	0.014	Length	0.109	Time	0.038
Length	0.007	Length	0.011	Style	0.088	Intention	0.032
Intention	0.005	Intention	0.009	Intention	0.008	Length	0.031
Vote	0.002	Vote	0.002	Vote	0.008	Vote	0.000
Non-text	0.269	Non-text	0.363	Non-text	0.313	Non-text	0.181
Text	0.063	Text	0.056	Text	0.258	Text	0.158

Realtor		**Reflectly**		**StockX**		**WeatherBug**	
Feature	IV	Feature	IV	Feature	IV	Feature	IV
Rating	0.163	Time	0.256	Time	0.196	Rating	0.282
Length	0.037	Sentiment	0.089	Length	0.170	Time	0.012
Sentiment	0.036	Style	0.068	Sentiment	0.040	Sentiment	0.002
Time	0.027	Length	0.051	Style	0.037	Intention	0.001
Style	0.018	Rating	0.050	Rating	0.006	Style	0.001
Intention	0.013	Intention	0.011	Intention	0.002	Length	0.000
Vote	0.000	Vote	0.000	Vote	0.002	Vote	0.000
Non-text	0.181	Non-text	0.310	Text	0.212	Non-text	0.347
Text	0.110	Text	0.185	Non-text	0.200	Text	0.008

univariate analysis which of course has the shortcoming of not considering multivariate interactions.

Due to the limited space, we present and discuss only the subset of key features (RQ2). For interpretability, we use a quantile-based discretization function to bin the values for features with continuous values. Otherwise, features are binned by ratio increments of 10%. The response rate for each feature is the percentage of all reviews having the feature that receive developer responses over all reviews having the feature.

We found that for the majority of apps, **reviews that are posted in the morning, during weekdays, and immediately after an app release are more likely to get developer responses.** This makes sense since developers should be more active during the day and on weekday as these are during business hours. A possible explanation as to why developers are more likely to respond to reviews posted shortly after a release is that developers want to address the concerns or issues reported soon after a release and are more active during that time. They may also be more likely to still be working on the first round of severe errors.

We observe that **reviews with lower ratings (3-star and below) are more likely to get developer responses** than reviews with higher ratings (4-star and 5-star reviews). This makes sense since developers may view such reviews as a great candidate to get users to change their low rating into a higher one. This finding is consistent with previous studies which found reviews with lower ratings to be more valuable to developers than reviews with higher ratings [9, 12].

We observe from Figure 3 that **reviews with longer bodies and titles have a greater likelihood of getting developer responses.** This is reasonable because longer review bodies and titles can hold more information and most reviews are short, which makes longer reviews stand out more. Interestingly, we observe that in some apps, the above statement only holds up to a certain length threshold. This may be due to developers discarding very long reviews.

We found that **reviews with higher use of positive words are less likely to get developer responses**. Likewise, for most apps, reviews with negative sentiment are more likely to get developer responses than reviews with neutral and positive sentiment.

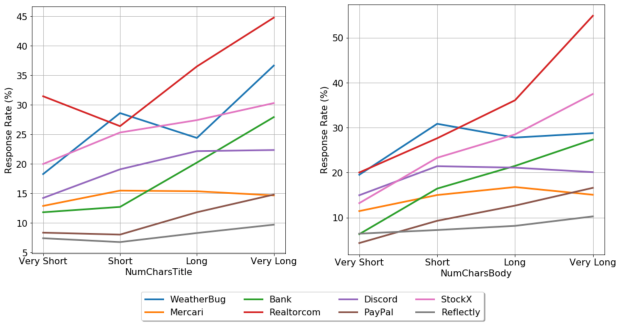


Figure 3: Effects of the *Length* features.

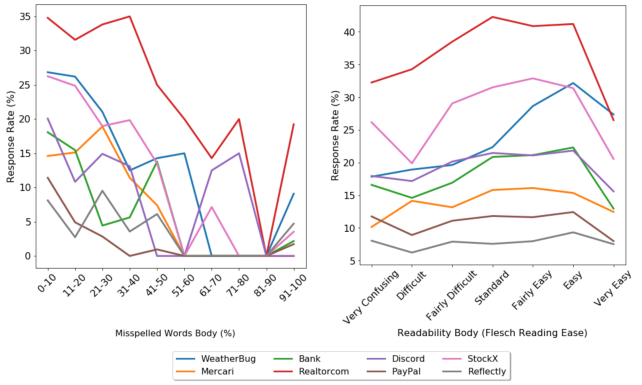


Figure 4: Effects of the *Style* features.

However, we observe that the sentiment feature (e.g., *Sentiment-Body*) does not seem to be a strong predictor which is also evident by the outcomes of our feature importance analysis in RQ2.

We observe that **reviews with the intention of reporting unexpected app behaviors (*ProblemDiscovery*) have the highest chance of receiving developer responses over reviews with other intentions**. For most apps, the likelihood of getting a developer response for reviews with the intention of seeking information (*InformationSeeking*) is higher than reviews that express ideas or suggests app improvement (*FeatureRequest*) and reviews with the intention of informing developers or other users about some aspect of the app (*InformationGiving*). However, the likelihood differences are not pronounced. Hence, these features provide weak predictive power (Table 4).

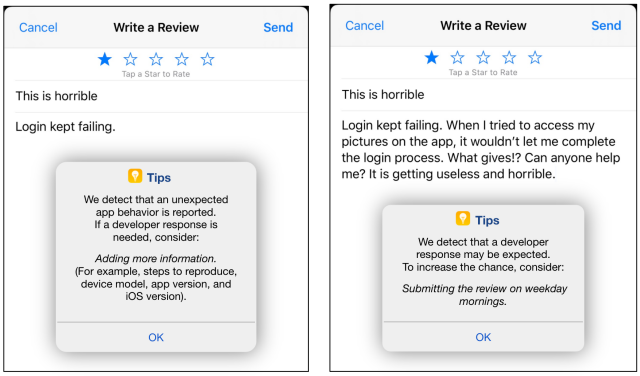
From Figure 4, we observe that **the higher the percentage of misspelled words a review has, the less likely it will receive a developer response.** Our observation is consistent with the outcomes of Schindler et al. [27], which stated that reviews with the greater use of negative style characteristics (e.g., spelling errors) were perceived as less valuable and may not be read. Similarly, from Figure 4, we observe that reviews with high readability score, according to the Flesch reading-ease test [5], are more likely to get a developer response than reviews with low readability score. This observation converges with Fang et al. [4] and Korfiatis et al. [13], which found that reviews that are easy to understand strongly correlate with high-value reviews. Interestingly, our result also suggests that the response likelihood drops-off significantly for reviews that are too easy to read. When we investigated reviews in this category, we found that they are mostly reviews with short sentences, consisting of one or two words (e.g., "Bad app").

6 IMPLICATIONS

The results of our work have several implications that could be of practical significance to a broad range of practitioners.

6.1 User-Side

The mock-up in Figure 5 shows a motivating example for a developer response prediction system that is integrated into the app store. Based on multiple features that can be extracted during review composition, the system determines whether the user is in need of a developer response, predicts the chance of getting the response, and suggests steps to increase that chance. To prevent users from exploiting this knowledge to elevate the priority of their otherwise mostly irrelevant reviews, the system may not reveal or suggest all highly weighted features to them, but only apply them on the developers' side when prioritizing reviews. As shown in Figure 5a, if it detects that the user is reporting an unexpected behavior of the app but only describing the bug in one short sentence, it will suggest the user increase the length by adding more relevant information or further clarifications, both of which should increase the chance of getting a developer response. Similarly, in Figure 5b, if it detects that the user is asking for help, it suggests to submit the review on weekday mornings to increase the chance of eliciting a developer response.



(a) **(b)**

Figure 5: A mock-up of a developer response prediction system that is integrated into the app store.

This suggestion system, therefore, would serve as a collection of evidence-based guidelines on how to provide the kind of feedback that will have a high chance of receiving a developer response. It is easy to see that, indirectly, this has a potential of raising the quality of user feedback on the app store by making it the interest of users to post focused, clear, and actionable reviews instead of rants.

6.2 Developer-Side

In this paper, we have identified several features that are important to developer response prediction. Based on our findings, researchers could develop tools that automatically prioritize reviews based on features or criteria that are corresponding to developers' actual response behavior (i.e., what they look for in a review) and not on intuitions or assumptions about their behavior. Table 4 shows that each app has its own individual feature importance ranking, suggesting that one-size-fits-all prioritization tools may not be the most effective. Therefore, the tools should also allow developers to calibrate by adjusting feature weights to their specific needs. Clearly, knowing which reviews to prioritize will let developers focus their resources and efforts optimally. Better response strategies should lead to increased user satisfaction and improved communication between users and developers.

7 THREATS TO VALIDITY

In this section, we have identified several threats to the validity and limitations of our study.

Completeness of Feature Set: Our results and conclusions were affected by the features we have selected and considered. This means that others employing a different sets of features and categorizing them differently may arrive at different research outcomes. The number of features possible is only limited by human imagination. Examples of features that we have considered for our prediction but could not incorporate yet are country of origin and the discussion topic. Praise could also be important in building relationships between developers and their users [20]. Addressing this and including more features is part of our plans for the future.

Tool and Method Reliability: While the tools we employed to extract features have been successfully integrated and used in related research and been shown to be effective in literature, it is still possible that inaccuracies exist in them that we did not take into account. In addition, since the existing tools are trained on review bodies, they may not be able to categorize the review titles well. We acknowledge that this may affect the results. Concerning our methods, we note that the *Time* feature may introduce a flaw due to the difficulty of pinpointing the actual location of a responding developer correctly. This is because developer teams may be distributed over any number of countries and time zones.

Developer responses: Understating features of user reviews that drive developers to respond to the reviews can be done in at least two ways. One possible way is to interview and survey developers. However, one flaw from this approach is that developers may not be aware of some or even all aspects of a review that compelled them to respond. In our study, we chose to investigate actual developer responses given to user reviews (i.e., an instance where developers provide a written response to a review). Therefore, we encourage future researchers to cross-reference our results through developer surveys.

Moreover, response strategies may vary considerably from app to app. While some apps may only have a sole developer who struggles to respond to user reviews, others might have a dedicated customer service team that responds to most reviews regardless of content. We acknowledge that this may affect our conclusions.

External validity: We acknowledge several limitations that may influence the generalizability of our findings. First, the number of our studied apps is limited and may not be representative of all apps in the iOS App Store. Thus, our data is subject to the App Sampling Problem [16]. However, we selected apps which vary along multiple dimensions such as category, owner, number of releases, age, and number of reviews and responses received.

Second, our app selection was biased towards top-ranked free apps. It is possible that analyzing paid or unpopular apps may yield different results.

Third, we only studied iOS apps. Hu et al. [10] have found that users have different reviewing cultures on different app stores. Additionally, some features are available on one app store, but not on the other, such as the review title. Therefore, we limit our study by focusing on only apps in one app store. In addition, the iOS App store separates reviews by their country of origin, while the Google Play Store separates reviews by languages. Previous studies have suggested that there exist cultural differences in how users from different countries provide app reviews [8, 29]. In order to protect our study from such cultural bias, we only studied US-based users' reviews. As a result, our findings may not hold true for apps on other app stores, such as the Google Play Store.

Fourth, we only experimented with one machine learning algorithm, the random forest algorithm. Therefore, we cannot claim that our results will apply to other algorithms. Hence, additional evaluation for different machine learning algorithms is needed.

Further study is still necessary to investigate whether our findings are valid for apps and reviews that do not share the same characteristics as those in our study. For reproducibility, the dataset and script used in this study are available at [28] and at the repository[9].

8 RELATED WORK

We summarize and group related work into two main research areas: user review analysis and developer response analysis.

8.1 User Review Analysis

While it is clear from previous work (e.g., Pagano et al. [20], Khalid et al. [12], and Palomba et al. [21]) that user reviews contain a wealth of information and have a great impact on an app's success, apps that are popular can receive thousands of reviews a day. For humans, reading such massive amounts of user reviews and extracting data from each one of them is impracticable. Therefore, during the past years, many researchers have put a lot of effort towards creating models to classify and summarize reviews to support developers in various software engineering activities. However, some of these models use criteria that are based on intuitions or assumptions about developers' actual behavior. For example, Gao et al. [6] only used rating and length to prioritize user reviews. Hence, our study use data-driven approach to understand what aspects of user reviews that incite developes' responses.

Chen et al. [2] proposed the *AR-Miner* tool which uses a machine learning algorithm to classify reviews as informative and non-informative. Guzman and Maalej [7] used *SentiStrength* and

[9] https://github.com/Kamonphop/ESEM20-Replication

topic modeling to extract sentiment of app features from user reviews. Similarly, Maalej and Nabil [15] introduced and evaluated several techniques to classify reviews into four types: bug reports, feature requests, user experience, and ratings. Panichella et al. [23] grouped 17 common topics in reviews identified by Pagano et al. [20] into 4 user review intentions that were relevant to developers. They then proposed a user review classifier called *ARdoc*. By using this classifier, Sorbo et al. [3] created the *SURF* tool that summarizes and classifies user reviews into common topic clusters, such as GUI and Pricing. Palomba et al. [22] used *ARdoc* for intention classification and a clustering algorithm to create the *ChangeAdvisor* tool which recommends changes to software artifacts. Villarroel et al. [33] developed a *CLAP* tool, an approach that uses a random forest algorithm to classify and uses a clustering algorithm to group user reviews based on the content in them. Gao et al. [6] utilized a topic modeling approach to detect emerging issues in reviews. Yet, none of the above tools gear towards prioritizing developer responses.

8.2 Developer Response Analysis

McIlroy et al. [18] analyzed review and response pairs from 10,713 top apps on the Google Play Store. They found that for only 13.8% of these apps, developers responded to reviews. They also found that users are more likely to increase their ratings after receiving responses. These outcomes are also supported by Hassan et al. [9]. Additionally, Hassan et al. investigated how features of reviews such as length and rating affect developer responses. However, they did not perform feature importance analysis and response prediction. Savarimuthu et al. [26] studied developer responses from a social norm perspective. They identified 12 norms in the responses and found, for example, that developers of 65% of the apps are aware of the personalization norms.

It is clear that less effort has been expended in studying the wide range of features of user reviews that prompt a developer's response. Hence, we take a step to fill this gap in research as we believe that understanding the key features that influence developers' response decision carries useful implications.

We published a preliminary results related to RQ2 in Srisopha et al. [30]. In comparison to this paper, we analyzed a larger sample of apps, added a new research question, and extended feature importance analysis to a group of features. In addition, we provided further insights and discussion to RQ2, extended the related work section, and provided implications and a motivating example.

9 CONCLUSIONS AND FUTURE WORK

In this paper, we identify features of user reviews that drive developers to respond to the reviews. We extract 7 different groups of features from user reviews: time, sentiment, rating, length, vote, intention, and writing style, and build models using the random forest algorithm to predict developer responses. Through a case study of eight popular free-to-download apps on the iOS App Store, we show the efficacy of the random forest algorithm in predicting developer responses using the extracted features. We achieve an average AUC of 0.84, outperforming all baselines by large margins. The findings indicate a potential application of applying feature engineering and machine learning to help developers automatically identify reviews they are most likely to respond to and focus on. In addition, we determine the relative importance of each individual feature and groups of features in predicting developer responses. We find that rating, review length, posted date and time, and the proportions of positive words are among the most important features to predict developer responses. We also uncover that non-text features are more important than text features. More importantly, we observe some variation in the rankings of important features which suggests that a one-size-fits-all prioritization tool or review ranking tool may not be suitable. Lastly, we provide insights into the relationships between the key features and developer responses. The results of our work could be of practical significance to a broad range of practitioners.

As our current research represents an initial effort to understand developer responses for the iOS App Store user reviews, we plan to expand the scope of our research to more features and more apps as well as including user reviews on the Google Play Store. To complete the prioritization of developers responses, we plan to study developers' response time and the relationships between review features and the response time.

REFERENCES

[1] Breiman, L. Random forests. *Machine learning 45*, 1 (2001), 5–32.

[2] Chen, N., Lin, J., Hoi, S. C., Xiao, X., and Zhang, B. Ar-miner: mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering* (2014), ACM, pp. 767–778.

[3] Di Sorbo, A., Panichella, S., Alexandru, C. V., Shimagaki, J., Visaggio, C. A., Canfora, G., and Gall, H. C. What would users change in my app? summarizing app reviews for recommending software changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (2016), ACM, pp. 499–510.

[4] Fang, B., Ye, Q., Kucukusta, D., and Law, R. Analysis of the perceived value of online tourism reviews: Influence of readability and reviewer characteristics. *Tourism Management 52* (2016), 498–506.

[5] Flesch, R. A new readability yardstick. *Journal of applied psychology 32*, 3 (1948), 221.

[6] Gao, C., Zeng, J., Lyu, M. R., and King, I. Online app review analysis for identifying emerging issues. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)* (2018), IEEE, pp. 48–58.

[7] Guzman, E., and Maalej, W. How do users like this feature? a fine grained sentiment analysis of app reviews. In *2014 IEEE 22nd international requirements engineering conference (RE)* (2014), IEEE, pp. 153–162.

[8] Guzman, E., Oliveira, L., Steiner, Y., Wagner, L. C., and Glinz, M. User feedback in the app store: a cross-cultural study. In *2018 IEEE/ACM 40th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)* (2018), IEEE, pp. 13–22.

[9] Hassan, S., Tantithamthavorn, C., Bezemer, C.-P., and Hassan, A. E. Studying the dialogue between users and developers of free apps in the google play store. *Empirical Software Engineering 23*, 3 (2018), 1275–1312.

[10] Hu, H., Bezemer, C.-P., and Hassan, A. E. Studying the consistency of star ratings and the complaints in 1 & 2-star user reviews for top free cross-platform android and ios apps. *Empirical Software Engineering* (2018), 1–34.

[11] Hu, M., and Liu, B. Mining and summarizing customer reviews. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining* (2004), ACM, pp. 168–177.

[12] Khalid, H., Shihab, E., Nagappan, M., and Hassan, A. E. What do mobile app users complain about? *IEEE Software 32*, 3 (2015), 70–77.

[13] Korfiatis, N., GarcíA-Bariocanal, E., and SáNchez-Alonso, S. Evaluating content quality and helpfulness of online product reviews: The interplay of review helpfulness vs. review content. *Electronic Commerce Research and Applications 11*, 3 (2012), 205–217.

[14] Kraskov, A., Stögbauer, H., and Grassberger, P. Estimating mutual information. *Physical review E 69*, 6 (2004), 066138.

[15] Maalej, W., and Nabil, H. Bug report, feature request, or simply praise? on automatically classifying app reviews. In *2015 IEEE 23rd international requirements engineering conference (RE)* (2015), IEEE, pp. 116–125.

[16] Martin, W., Harman, M., Jia, Y., Sarro, F., and Zhang, Y. The app sampling problem for app store mining. In *Proceedings of the 12th Working Conference on Mining Software Repositories* (2015), IEEE Press, pp. 123–133.

[17] Martin, W., Sarro, F., Jia, Y., Zhang, Y., and Harman, M. A survey of app store analysis for software engineering. *IEEE transactions on software engineering*

43, 9 (2016), 817–847.

[18] McIlroy, S., Shang, W., Ali, N., and Hassan, A. E. Is it worth responding to reviews? studying the top free apps in google play. *IEEE Software 34*, 3 (2015), 64–71.

[19] Omondiagbe, O. P., Licorish, S. A., and MacDonell, S. G. Features that predict the acceptability of java and javascript answers on stack overflow. In *Proceedings of the Evaluation and Assessment on Software Engineering* (2019), ACM.

[20] Pagano, D., and Maalej, W. User feedback in the appstore: An empirical study. In *2013 21st IEEE international requirements engineering conference (RE)* (2013), IEEE, pp. 125–134.

[21] Palomba, F., Linares-Vásquez, M., Bavota, G., Oliveto, R., Di Penta, M., Poshyvanyk, D., and De Lucia, A. Crowdsourcing user reviews to support the evolution of mobile apps. *Journal of Systems and Software 137* (2018), 143–162.

[22] Palomba, F., Salza, P., Ciurumelea, A., Panichella, S., Gall, H., Ferrucci, F., and De Lucia, A. Recommending and localizing change requests for mobile apps based on user reviews. In *Proceedings of the 39th international conference on software engineering* (2017), IEEE Press, pp. 106–117.

[23] Panichella, S., Di Sorbo, A., Guzman, E., Visaggio, C. A., Canfora, G., and Gall, H. C. How can i improve my app? classifying user reviews for software maintenance and evolution. In *2015 IEEE international conference on software maintenance and evolution (ICSME)* (2015), IEEE, pp. 281–290.

[24] Porter, M. F. An algorithm for suffix stripping. *program 14*, 3 (1980), 130–137.

[25] Salehan, M., and Kim, D. J. Predicting the performance of online consumer reviews: A sentiment mining approach to big data analytics. *Decision Support Systems 81* (2016), 30–40.

[26] Savarimuthu, B., Licorish, S., Devananda, M., Greenheld, G., Dignum, V., and Dignum, F. Developers' responses to app review feedback-a study of communication norms in app development. In *Proc. of the COIN 2017 workshop@ AAMAS (2017)* (2017).

[27] Schindler, R. M., and Bickart, B. Perceived helpfulness of online consumer reviews: The role of message content and style. *Journal of Consumer Behaviour 11*, 3 (2012), 234–243.

[28] Srisopha, K., Link, D., Swami, D., and Boehm, B. A Replication Package of Learning Features that Predict Developer Responses for iOS App Store Reviews. https://doi.org/10.5281/zenodo.3960965.

[29] Srisopha, K., Phonsom, C., Lin, K., and Boehm, B. Same app, different countries: A preliminary user reviews study on most downloaded ios apps. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2019), IEEE, pp. 76–80.

[30] Srisopha, K., Swami, D., Link, D., and Boehm, B. How features in ios app store reviews can predict developer responses. In *Proceedings of the Evaluation and Assessment in Software Engineering* (2020), pp. 336–341.

[31] Strobl, C., Boulesteix, A.-L., Kneib, T., Augustin, T., and Zeileis, A. Conditional variable importance for random forests. *BMC bioinformatics 9*, 1 (2008), 307.

[32] Thelwall, M., Buckley, K., Paltoglou, G., Cai, D., and Kappas, A. Sentiment strength detection in short informal text. *Journal of the American Society for Information Science and Technology 61*, 12 (2010), 2544–2558.

[33] Villarroel, L., Bavota, G., Russo, B., Oliveto, R., and Di Penta, M. Release planning of mobile apps based on user reviews. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)* (2016), IEEE, pp. 14–24.

[34] Yang, L., Dumais, S. T., Bennett, P. N., and Awadallah, A. H. Characterizing and predicting enterprise email reply behavior. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval* (2017), ACM, pp. 235–244.